

# Software-Projekt I

Prof. Dr. Rainer Koschke

Arbeitsgruppe Softwaretechnik  
Fachbereich Mathematik und Informatik  
Universität Bremen

Sommersemester 2013

# Objektorientierte Modellierung I

## 1 Objektorientierte Modellierung

- Lernziele
- Modellbildung
- Objektorientierte Modellierung
- Geschäftsprozesse
- Anwendungsfälle
- Ermittlung von Anwendungsfällen
- Klassendiagramme
- Schnittstellen
- Paketdiagramme
- Verhaltenseigenschaften
- Aktivitätsdiagramme
- Interaktionsdiagramme
- Sequenzdiagramme
- Kommunikationsdiagramme

# Objektorientierte Modellierung II

- Zustandsautomatendiagramme



- Was ist Modellierung und wozu brauchen wir sie?
- Wie wird objektorientiert modelliert?
- Wie kann man die Unified Modeling Language (UML) für die Modellierung verwenden?

Anmerkung: kein umfassender UML-Kurs; zur UML siehe z.B.: Störle (2005); Rupp u. a. (2007).

## Modelle und Wirklichkeit



*Ceci n'est pas une pipe.*

# Modellierung

- Was ist ein Modell?
  - Abbild eines Originals
- Wozu modellieren wir?
  - um etwas zu verstehen
  - um Vorhersagen machen zu können
  - um etwas zu dokumentieren
- Wann modellieren wir?
  - jederzeit: Projektplan, Anforderungen, Architektur, ...

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle



# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen zwischen Objekten
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen zwischen Objekten
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Multiplizitäten der Assoziationen
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen zwischen Objekten
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen



# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - ▶ Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - ▶ Identifiziere Verhalten der Objekte

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - ▶ Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - ▶ Identifiziere Verhalten der Objekte
  - ▶ Beschreibe das Verhalten (Vor- und Nachbedingungen)

## Ist $\rightarrow$ Soll

Schwierig: Dinge im Abstrakten beschreiben.

Einfacher: von konkreten Geschäftsprozessen ausgehen.

### Definition

Ein **Geschäftsprozess** ist eine Folge von Schritten oder ein Rezept, um ein Geschäftsergebnis zu erzielen.

# Ist → Soll

Schwierig: Dinge im Abstrakten beschreiben.

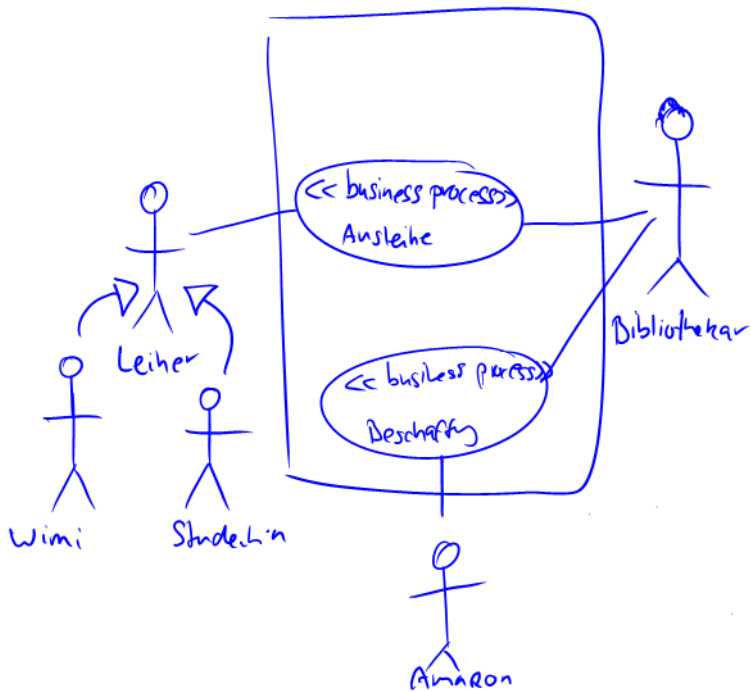
Einfacher: von konkreten Geschäftsprozessen ausgehen.

## Definition

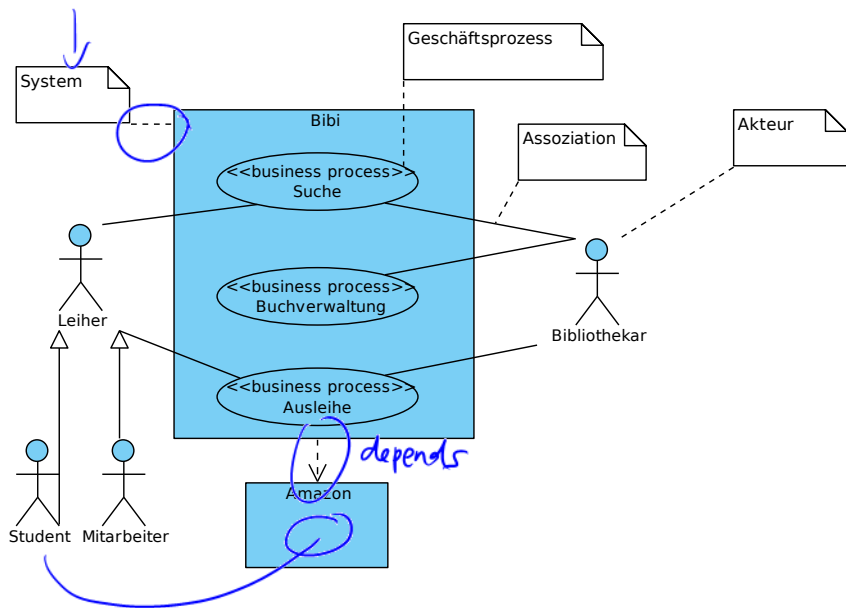
Ein **Geschäftsprozess** ist eine Folge von Schritten oder ein Rezept, um ein Geschäftsergebnis zu erzielen.

Beispiele:

- Ausleihe: Vormerkung, Abholung, Mahnung, Rückgabe
- Buchverwaltung: Anschaffung neuer Bücher, Ausinventarisierung
- Buchsuche



# Geschäftsprozesse in UML (OMG)



# Akteur

## Definition

### **Akteur**

- repräsentiert eine kohärente Menge von Rollen, die von Benutzern in der Interaktion mit dem System eingenommen werden können
- können Menschen und andere Dinge sein (z.B. andere automatisierte Systeme)

# Akteur

## Definition

### Akteur

- repräsentiert eine kohärente Menge von Rollen, die von Benutzern in der Interaktion mit dem System eingenommen werden können
- können Menschen und andere Dinge sein (z.B. andere automatisierte Systeme)

Beispiel rechnergestützter Geschäftsprozess *Ausleihe* in der Bibliothek:

- Buchsuche: Mitarbeiter sucht ein Buch
  - Akteure: Mitarbeiter, System
- Abholung: Bibliothekar händigt Buch aus und vermerkt die Ausleihe im System
  - Akteure: Bibliothekar, System



# Geschäftsprozess und Anwendungsfall (Use-Case)

Merkmale von Geschäftsprozessen:

- systemübergreifend,
- unterbrechbar,
- lang laufend,
- erfordern fortlaufende Interaktion zwischen vielen Akteuren,
- bestehen aus Anwendungsfällen.

# Geschäftsprozess und Anwendungsfall (Use-Case)

Merkmale von Geschäftsprozessen:

- systemübergreifend,
- unterbrechbar,
- lang laufend,
- erfordern fortlaufende Interaktion zwischen vielen Akteuren,
- bestehen aus Anwendungsfällen.

## Definition

### **Anwendungsfall (auch: Nutzfall)**

- beschreibt eine Menge von Aktionssequenzen (Varianten eingeschlossen)
- jede Sequenz repräsentiert die Interaktion zwischen externen Akteuren mit dem System
- Folge ist beobachtbares Resultat, relevant für Akteur

# Beispiel Bibliotheksverwaltung



# Textuelle Beschreibung von Anwendungsfällen I

- Name: Vormerkung eines Buches
- Akteure:
  - ▶ Leiher
- Vorbedingung:
  - ▶ Leiher möchte Buch ausleihen
  - ▶ Leiher hat ein Konto in der Bibliothek
  - ▶ Rechner von Leiher hat Verbindung zu Bibliotheks-Server
- Nachbedingung:
  - ▶ Leiher hat Buch vorgemerkt
- Ablauf:
  - 1 Mitarbeiter startet Client und meldet sich beim Bibliotheks-Server an
  - 2 Leiher gibt Suchkriterium an (z.B. Name des Autoren)
  - 3 Server liefert alle passenden Bücher
  - 4 Mitarbeiter wählt Buch aus und markiert es zur Vormerkung

# Textuelle Beschreibung von Anwendungsfällen II

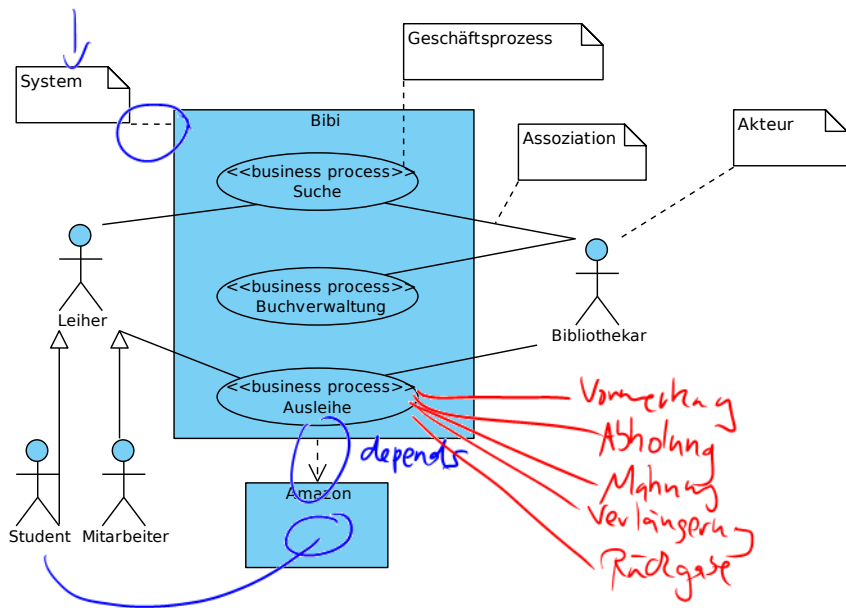
- Varianten:

- ▶ Leihher findet kein passendes Buch
  - Leihher verändert Suchkriterium bzw. gibt Suche auf
- ▶ Leihher hat Passwort vergessen
  - System schickt das Passwort an abgespeicherte E-Mail-Adresse
- ▶ Buch ist bereits mehrfach vorgemerkt
  - Leihher verzichtet auf Vormerkung
- ▶ keine Verbindung zwischen Client und Server
  - Leihher versucht es später wieder

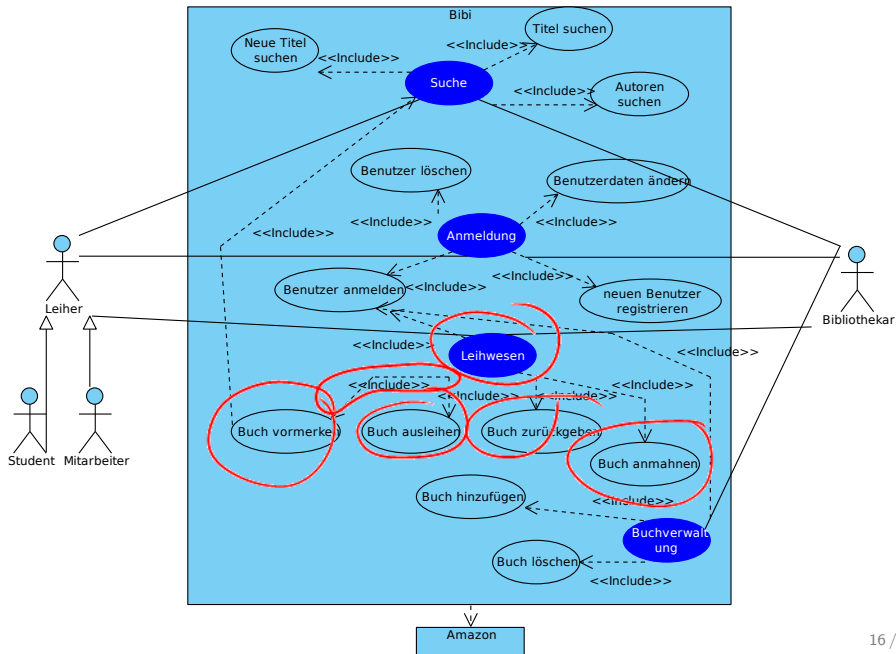
# Geschäftsprozesse, Anwendungsfälle und Akteure

- 1 Bestimme Geschäftsprozesse
- 2 Identifiziere Akteure
- 3 Betrachte System aus der Sicht der Akteure
- 4 Bestimme Anwendungsfälle für Akteure
  - liefert möglicherweise neue Akteure
- 5 zurück zu 2, bis keine neuen Akteure/Anwendungsfälle mehr gefunden werden können
- 6 identifiziere gemeinsame Anteile in Anwendungsfällen und faktorisiere entsprechend
- 7 fasse ähnliche Anwendungsfälle und Akteure in Vererbungshierarchien zusammen

# Geschäftsprozesse in UML (OMG)

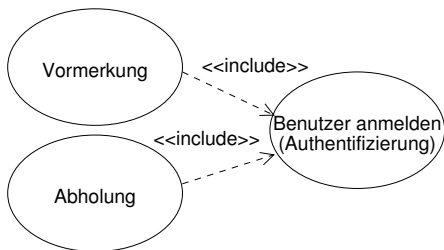


# UML-Notation für Anwendungsfälle (OMG)

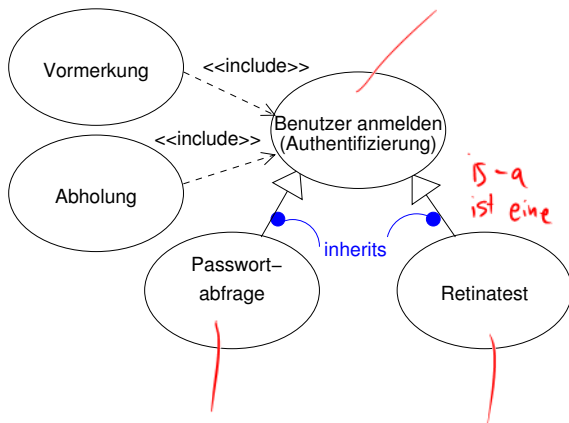




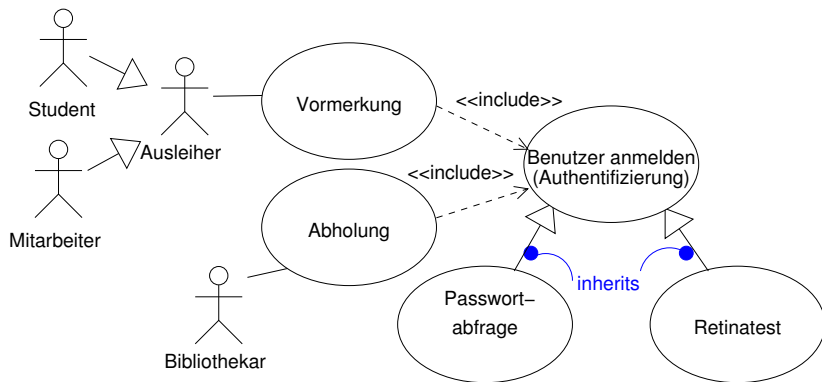
# Strukturierungskonzepte für Anwendungsfälle (OMG)



# Strukturierungskonzepte für Anwendungsfälle (OMG)



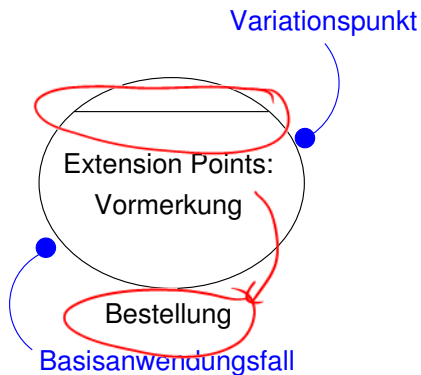
# Strukturierungskonzepte für Anwendungsfälle (OMG)



A hand-drawn, irregular oval shape with a black outline, centered on the page. The word "Bestellung" is written inside it.

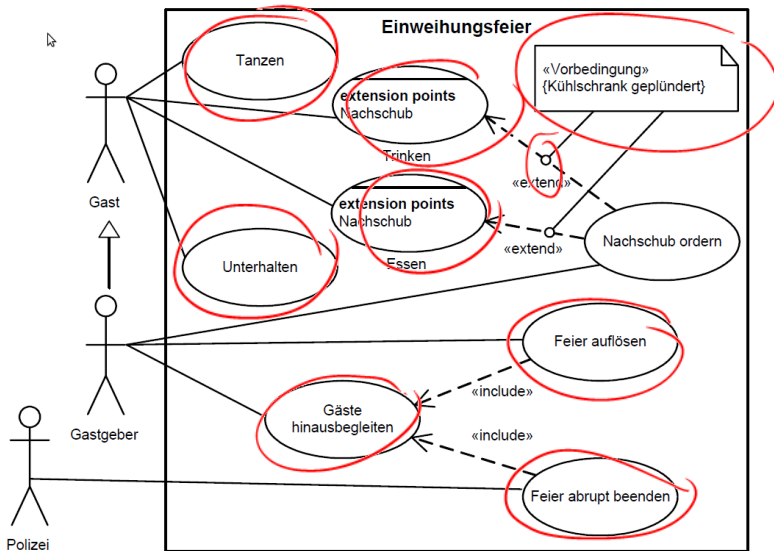
Bestellung

# Strukturierungskonzepte für Anwendungsfälle (OMG)





# Beispiel: Hauseinweihung



# Beschreibung von Anwendungsfällen

## Anwendungsfalldiagramme:

- deklarieren voneinander unabhängige Anwendungsfälle
- beschreiben die Einbettung der Anwendungsfälle in den Systemkontext (externe Akteure)
- beschreiben (konkretisieren) die Anwendungsfälle jedoch nicht
  - folgt später
- sind Ausgangspunkt für die objektorientierte Modellierung
  - ▶ statische Eigenschaften (Attribute)
  - ▶ dynamische Eigenschaften (Verhalten)



# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - ▶ Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - ▶ Identifiziere Verhalten der Objekte
  - ▶ Beschreibe das Verhalten (Vor- und Nachbedingungen)

# Beispiel Bibliothek

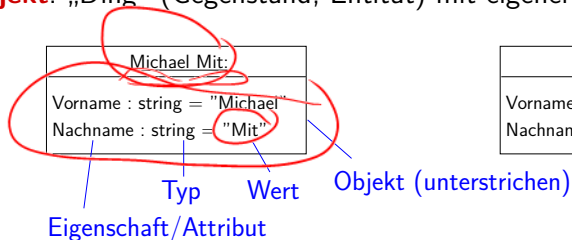
Identifiziere Objekte: Suche nach Substantiven in Anwendungsfällen.

## Operation: Vormerkung und Abholung

- Mitarbeiter Michel Mit möchte Buch ausleihen
- Michel Mit sucht online nach einem Buch mit Titel Softwaretechnik
- Michel Mit reserviert das gefundene Buch
- Michel Mit geht zum Bibliothekar Bernd Bib der Universität Bremen
- Bernd Bib sucht nach Reservierung
- Bernd Bib hält Ausleihe fest
- Michel Mit geht mit Buch von dannen

# Anwendungsfall → Objekte

**Objekt:** „Ding“ (Gegenstand, Entität) mit eigener Identität



<u>Bernd Bib:</u>
Vorname : string = "Bernd"
Nachname : string = "Bib"

<u>Softwaretechnik:</u>
Autoren : string = "Ian Sommerville"
Titel : string = "Titel"
Verlag : string = "Pearson Education"
Erscheinungsjahr : int = 2008

<u>Uni Bremen:</u>
Adresse : string = "Am Fallturm 1, 2.57"
Name : string = "Handapparat AG ST"

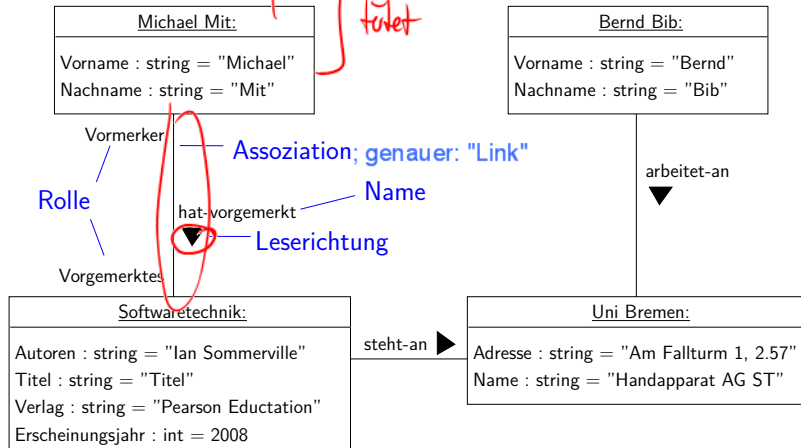
# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - **Bestimme Assoziationen zwischen Objekten**
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Multiplizitäten der Assoziationen
  - Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)

# Anwendungsfall → Objekte

**Assoziation:** Beziehung zwischen Objekten



hat-vorgemerkt (Michael Mit, Softwaretechnik)  
vormerker vorgemerkt

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - ▶ Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - ▶ Identifiziere Verhalten der Objekte
  - ▶ Beschreibe das Verhalten (Vor- und Nachbedingungen)

# Objekte versus Klassen

## Instanz-Ebene

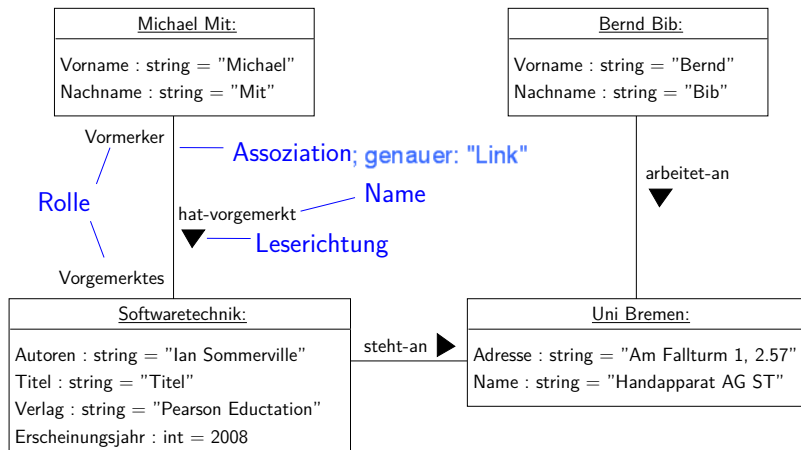
- **Objektdiagramme** beschreiben statische Zusammenhänge auf der Ebene einzelner, konkreter Dinge

## Schema-Ebene

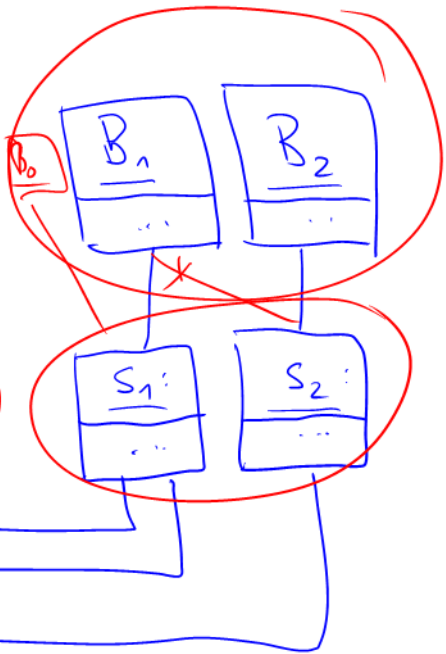
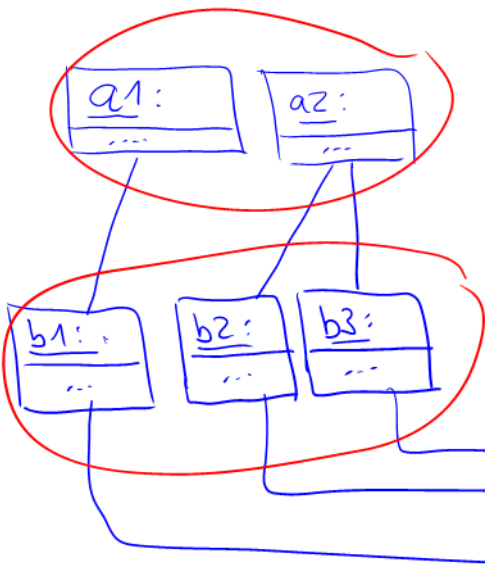
- **Klassendiagramme** beschreiben statische Zusammenhänge unabhängig von Details konkreter Objekte, auf der Ebene mehrerer gleichartiger Dinge

# Anwendungsfall → Objekte

**Assoziation:** Beziehung zwischen Objekten

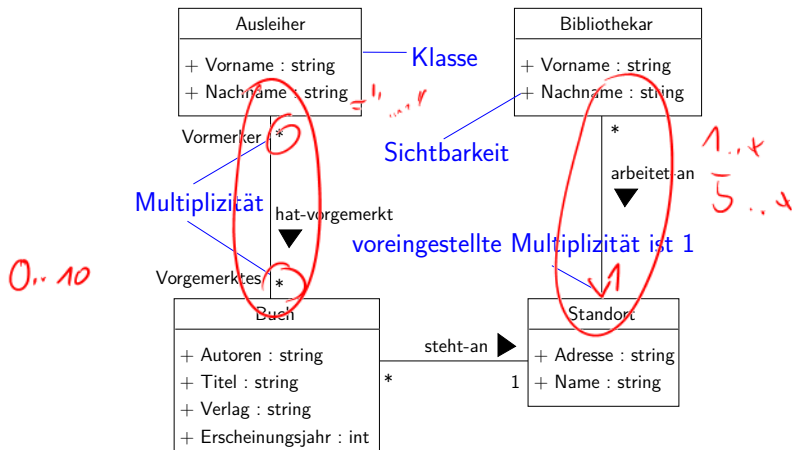






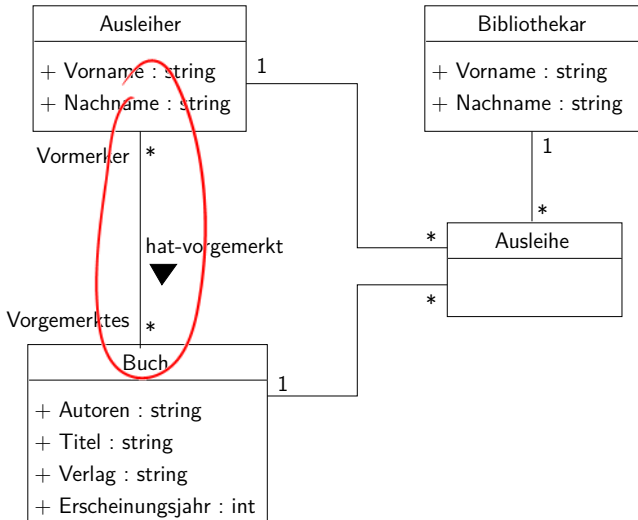
# Objekte → Klassen

**Klasse:** Menge von gleichartigen Objekten mit gemeinsamen Eigenschaften



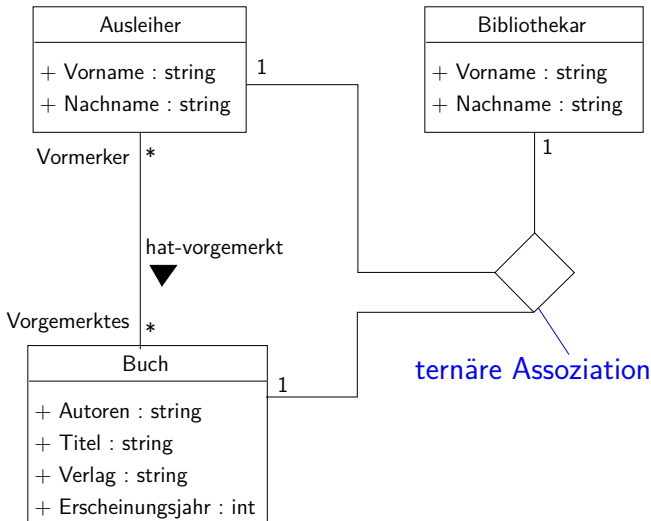
# Objekte → Klassen: Assoziationen

mehrstellige Assoziation als Klasse:



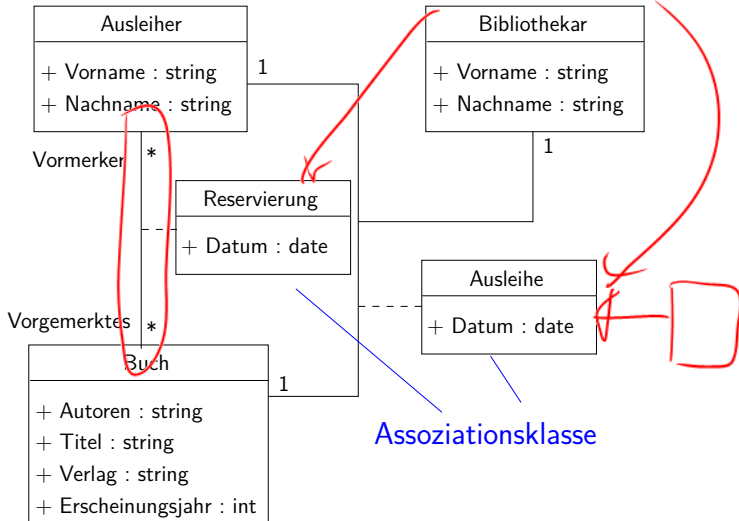
# Objekte → Klassen: Assoziationen

mehrstellige Assoziation:



# Objekte → Klassen: Assoziationen

(mehrstellige) Assoziation mit Attributen als Assoziationsklasse:

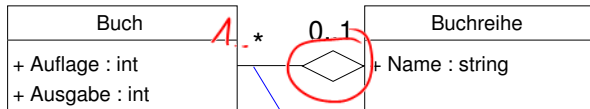


# Aufbau von Objekten

## Definition

### Aggregation

- ist spezielle Assoziation zur Verdeutlichung von „Teil-Ganzes-Beziehungen“
- beschreibt Zusammenfassung von Komponenten zu einem Aggregat



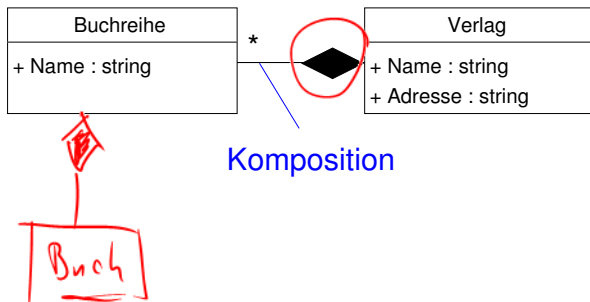
Aggregation

# Aufbau von Objekten

## Definition

**Komposition** ist spezielle Aggregation:

- Existenz der Komponenten ist an die Existenz des Aggregats gekoppelt,
- jede Komponente gehört zu ~~genau~~ **zu einem Zeitpunkt höchstens** einem Aggregat (strong ownership)



# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - Identifiziere Objekte
  - Identifiziere Eigenschaften der Objekte
  - Bestimme Assoziationen zwischen Objekten
  - Fasse Objekte zu Klassen zusammen
  - Bestimme Multiplizitäten der Assoziationen
  - **Ordne Klassen in Vererbungshierarchien ein**
- Erstelle Verhaltensmodell
  - Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - Identifiziere Verhalten der Objekte
  - Beschreibe das Verhalten (Vor- und Nachbedingungen)



# Generalisierungen

- sind spezielle Beziehung auf Schema-Ebene
- beschreiben Beziehungen zwischen einer allgemeineren Klasse (Oberklasse, Superclass) und einer spezielleren Klasse (Unterklasse, Subclass)
- die Unterklasse „erbt“ die Eigenschaften der Oberklasse:
  - ▶ Attribute
  - ▶ Methoden und deren Aufrufchnittstellen
  - ▶ Assoziationen

# Generalisierungen

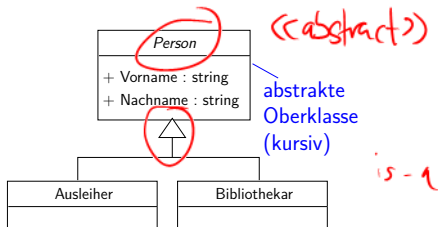
- sind spezielle Beziehung auf Schema-Ebene
- beschreiben Beziehungen zwischen einer allgemeineren Klasse (Oberklasse, Superclass) und einer spezielleren Klasse (Unterklasse, Subclass)
- die Unterklasse „erbt“ die Eigenschaften der Oberklasse:
  - ▶ Attribute
  - ▶ Methoden und deren Aufrufschnittstellen
  - ▶ Assoziationen



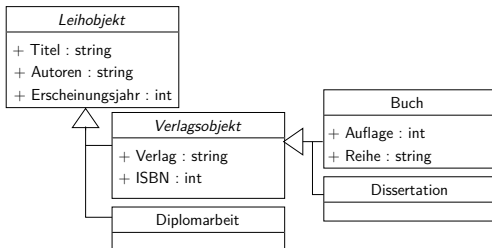
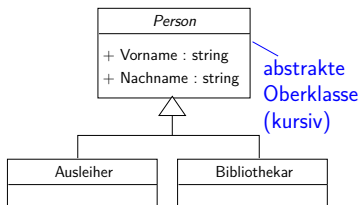
Unterklassen können

- neue Attribute, Methoden und Assoziationen definieren
- ererbte Attribute, Methoden und Assoziationen redefinieren (überschreiben)
- von mehreren Oberklassen erben (Mehrfachvererbung)

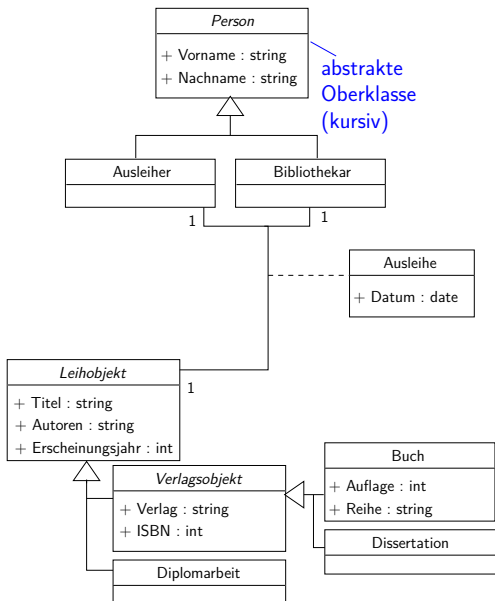
# Klassen → Klassenhierarchien



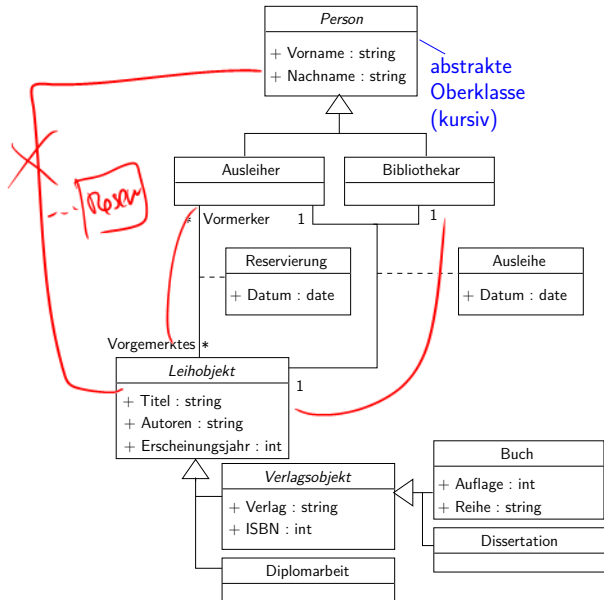
# Klassen → Klassenhierarchien



# Klassen → Klassenhierarchien



# Klassen → Klassenhierarchien



## Liskovs Substitutionsprinzip (1988; 1994)

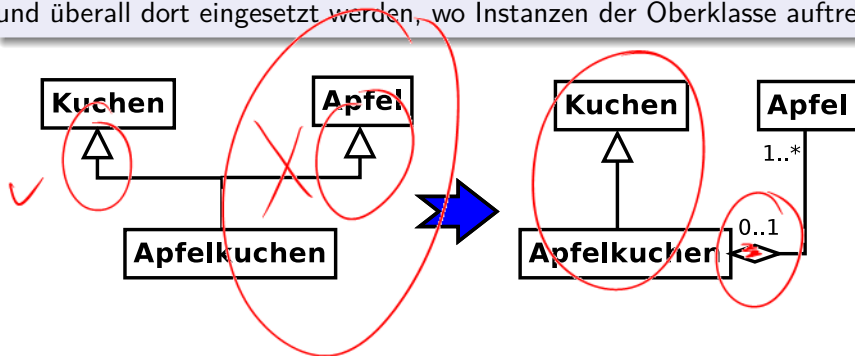
Wie kann man entscheiden, ob eine Klasse eine Spezialisierung einer anderen Klasse ist?

# Liskovs Substitutionsprinzip (1988; 1994)

Wie kann man entscheiden, ob eine Klasse eine Spezialisierung einer anderen Klasse ist?

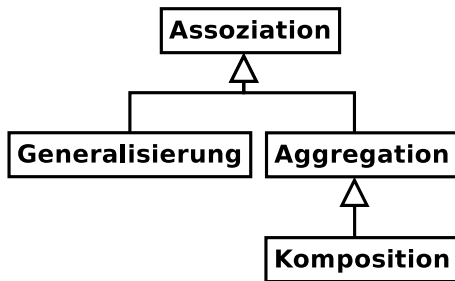
## Definition

**Liskovs Substitutionsprinzip:** jede Instanz der Unterklasse kann immer und überall dort eingesetzt werden, wo Instanzen der Oberklasse auftreten.





# Vergleich der Assoziationstypen



- Vererbung: ist-ein-Relation (Liskovs Substitutionsprinzip erfüllt)
- Aggregation: teil-von-Relation
- Komposition: teil-von-Relation
  - Teil gehört zu genau einem Ganzen
  - Teil existiert nur im Kontext des Ganzen
- allgemeine Assoziation: sonst

# Zusammenfassung Klassendiagramme

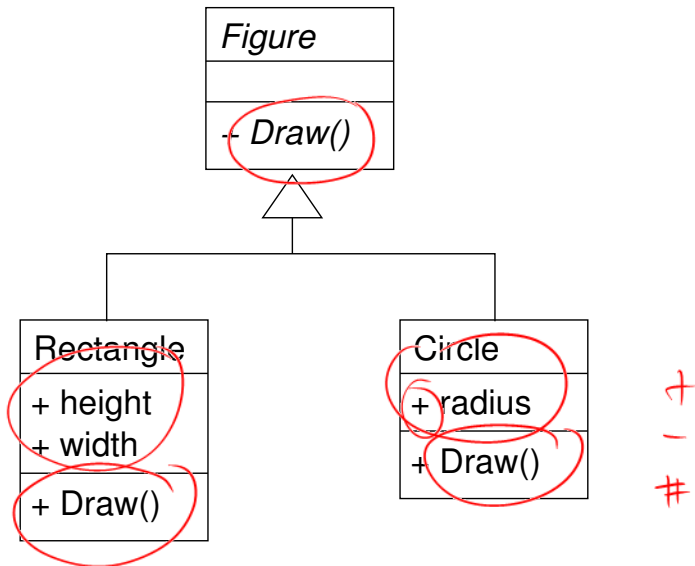
## Beschreibungsinhalt

- statische Systemaspekte
- Beschreibung der wesentlichen, unterscheidbaren Dinge eines Systems und ihrer Beziehungen

## zentrale Modellierungskonstrukte:

- Klassen
- Assoziationen (Beziehungsklassen)
  - spezielle Assoziationen: Generalisierung und Aggregation/Komposition

# Klassendiagramme im Entwurf (statt Datenmodellierung)



# Eine offene Schnittstelle

```
abstract class Figure {  
    public abstract void draw ();  
}
```

```
class Circle extends Figure {  
    public void draw () {};  
    public int radius;  
}
```

```
class Rectangle extends Figure {  
    public void draw () {};  
    public int height;  
    public int width;  
}
```

# Geheimnisprinzip (Information Hiding) nach Parnas (1972)

Schnittstellen sind ein Kontrakt zwischen:

- Verwender:
  - darf sich nur auf zugesicherte Annahmen verlassen
  - muss Vorbedingungen einhalten
- Anbieter:
  - muss zugesichertes Verhalten implementieren
  - darf sich nur auf zugesicherte Vorbedingungen verlassen

# Geheimnisprinzip (Information Hiding) nach Parnas (1972)

Schnittstellen sind ein Kontrakt zwischen:

- Verwender:
  - darf sich nur auf zugesicherte Annahmen verlassen
  - muss Vorbedingungen einhalten
- Anbieter:
  - muss zugesichertes Verhalten implementieren
  - darf sich nur auf zugesicherte Vorbedingungen verlassen

Der Kontrakt führt zu einer Kopplung zwischen Verwender und Anbieter.

Schnittstellen werden so entworfen, dass

- Kopplung auf das Mindestmaß beschränkt wird;
- d.h. die Details, die sich ändern können, werden hinter Schnittstelle verborgen.

# Programmiersprachenunterstützung

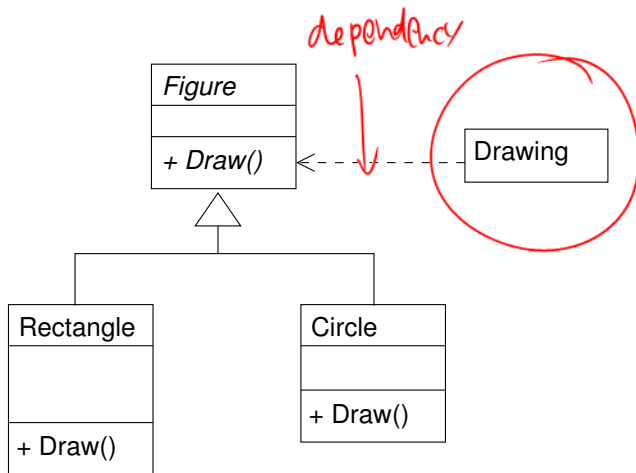
```
abstract class Figure {  
  
    public abstract void draw ();  
}  
class Circle extends Figure {  
  
    public void draw () {};  
    private int radius;  
}  
class Rectangle extends Figure {  
  
    public void draw () {};  
    private int height;  
    private int width;  
}
```

## Verwender der Klasse

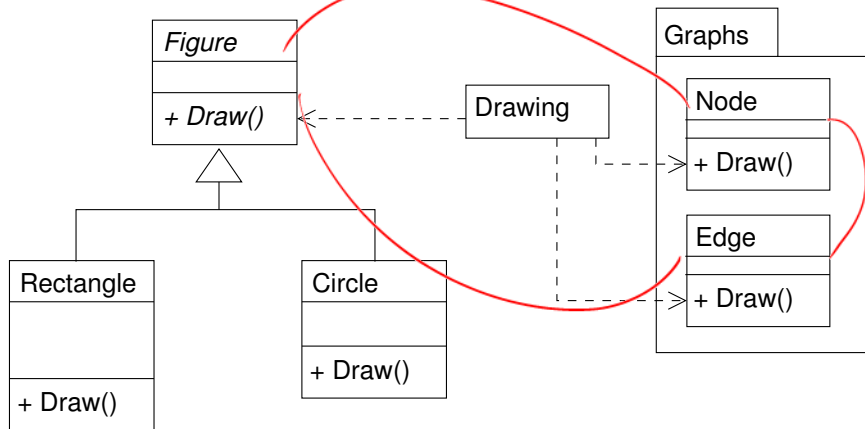
```
public void Drawing (Figure aFigure, int times) {  
    for (int i = 0; i < times; i++) {  
        aFigure.draw ();  
    }  
}
```



# Klassendiagramm

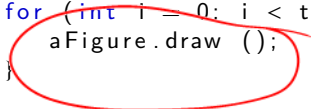


# Klassendiagramm

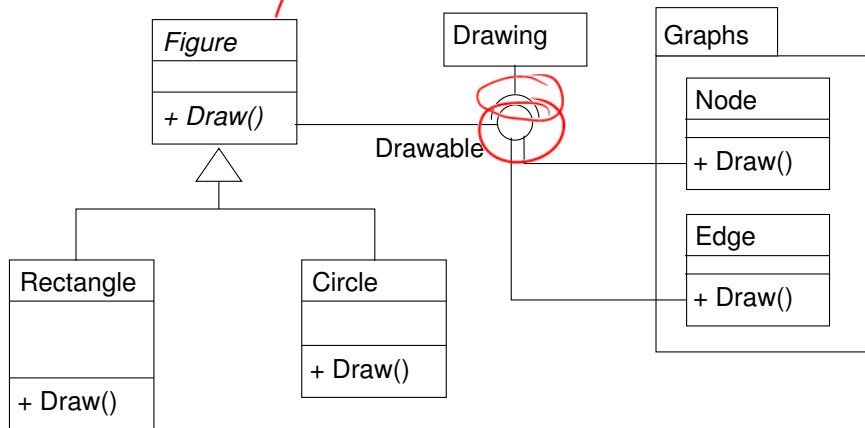


## Verwender der Klasse

```
public void Drawing (Figure aFigure, int times) {  
    for (int i = 0; i < times; i++) {  
        aFigure.draw ();  
    }  
}
```



# Klassendiagramm



# Schnittstelle als eigenes Sprachkonstrukt

Schnittstelle:

```
interface Drawable {  
    void draw ();  
}
```

# Schnittstelle als eigenes Sprachkonstrukt

Schnittstelle:

```
interface Drawable {  
    void draw ();  
}
```

Schnittstellenverwender:

```
public void Drawing (Drawable drawableObject , int times) {  
    for (int i = 0; i < times; i++) {  
        drawableObject.draw ();  
    }  
}
```

# Schnittstelle als eigenes Sprachkonstrukt

Schnittstelle:

```
interface Drawable {  
    void draw ();  
}
```

Schnittstellenverwender:

```
public void Drawing (Drawable drawableObject , int times) {  
    for (int i = 0; i < times; i++) {  
        drawableObject.draw ();  
    }  
}
```

Schnittstellenanbieter:

```
abstract class Figure implements Drawable {}
```

*Handwritten note:* implements "Vide implements"

# Zusammenfassung zu Schnittstellen

- Schnittstelle ist Kontrakt zwischen Anbieter und Verwender, der die erlaubten wechselseitigen Annahmen festlegt
- Programmiersprachen erlauben die Spezifikation syntaktischer Eigenschaften von Schnittstellen
- moderne Programmiersprachen bieten Schnittstellen als eigenes Sprachkonstrukt an
- nur wenige erlauben die Spezifikation semantischer Eigenschaften
  - Vor- und Nachbedingungen
  - weitere Zusicherungen, wie z.B. Speicher- und Zeitkomplexität



# Paketdiagramme

Große Modelle benötigen Strukturierungen

## Paketdiagramme

Zusammenfassung von Modellelementen zu Gruppen.

# Paketdiagramme

Große Modelle benötigen Strukturierungen

## Paketdiagramme

Zusammenfassung von Modellelementen zu Gruppen.

Anwendbarkeit:

- alle Arten von Modellelementen
- häufig für
  - Anwendungsfälle
  - Klassen (Programmiersprachenunterstützung durch Packages oder Namespaces)

# Paketdiagramme

Große Modelle benötigen Strukturierungen

## Paketdiagramme

Zusammenfassung von Modellelementen zu Gruppen.

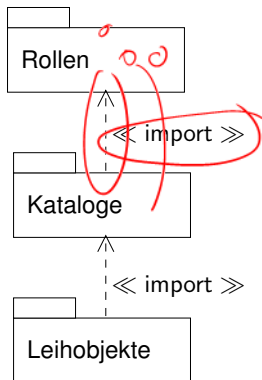
Anwendbarkeit:

- alle Arten von Modellelementen
- häufig für
  - Anwendungsfälle
  - Klassen (Programmiersprachenunterstützung durch Packages oder Namespaces)

Kriterien:

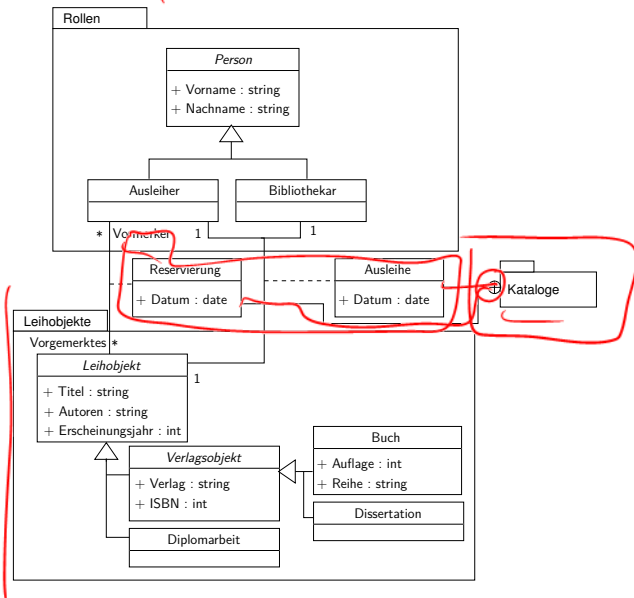
- logisch zusammengehörige Elemente
- $7 \pm 3$ -Regel

# Paketdiagramme



`<< import >>`: importierendes Paket fügt Namen des importierten Pakets zu eigenen Namen hinzu

# Paketdiagramme: Schachtelung



# Resultat der statischen Modellierung

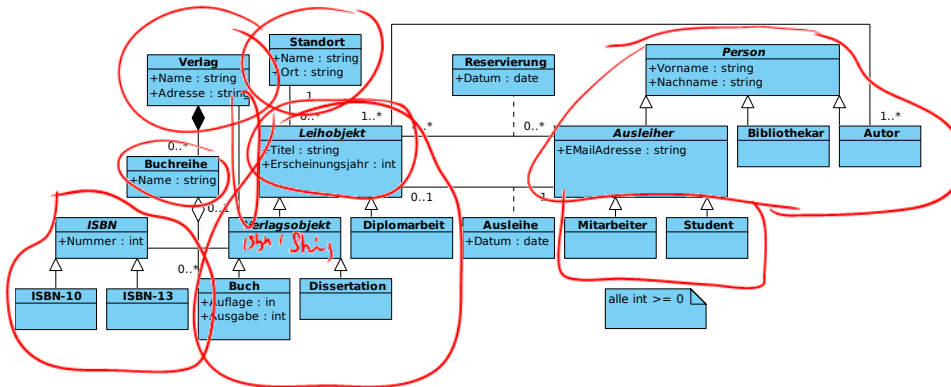
## Definition

**Statisches Datenmodell:** beschreibt die statischen Konzepte und ihre Beziehungen.

- Domänenmodellierung: Konzepte stammen aus Anwendungsdomäne
- Entwurfsmodellierung: Konzepte sind Datenstrukturen für die Konzepte der Anwendungsdomäne und Konzepte aus der Programmierdomäne (z.B. Listen, Hashtabellen etc.)

# Resultat der statischen Anforderungsanalyse im Beispiel

Datenmodell der Anwendungsdomäne:




# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

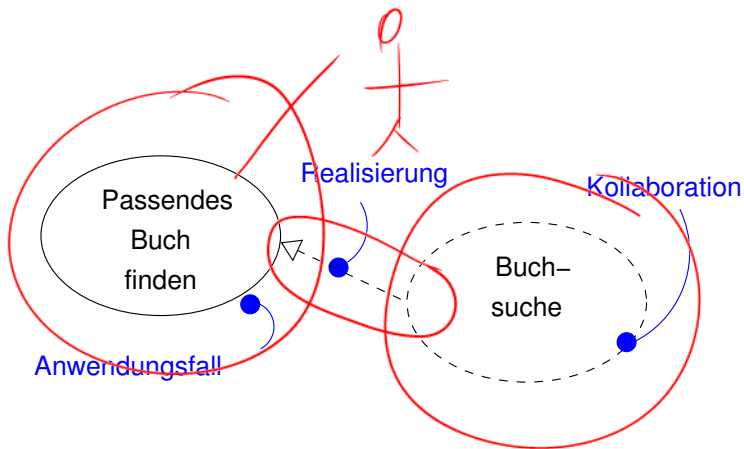
- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- **Erstelle Verhaltensmodell**
  - ▶ **Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen**
  - ▶ Identifiziere Verhalten der Objekte
  - ▶ Beschreibe das Verhalten (Vor- und Nachbedingungen)



# Beschreibung von Anwendungsfällen

- textuelle Szenario-Beschreibungen (siehe oben)
- Aktivitätsdiagramme
- Interaktionsdiagramme 
- Zustandsdiagramme

# UML-Notation für Anwendungsfälle (OMG)



# UML-Interaktionsdiagramme (OMG)

Beschreibungsinhalt:

- dynamische Systemaspekte
- exemplarische Beschreibung von Interaktionsabfolgen zwischen kollaborierenden Objekten (Inter-Objektverhalten)

zentrale Modellierungskonstrukte:

- Objekte: „Dinge“ mit eigener Identität
- Nachrichten
  - ▶ beschreiben den Austausch von Informationen zwischen Objekten zum Auslösen von Aktivitäten
  - ▶ sind Signale/Ereignisse oder Methodenaufrufe

# UML-Interaktionsdiagramme (OMG)

## Anwendung

- Präzisierung von Szenarien (exemplarische Folgen von Aktivitäten)
- Protokollierung des Nachrichtenaustausches
- Erhebung der von einzelnen Objekten bereit gestellten Funktionalität (Dienste, Methoden)

## Varianten

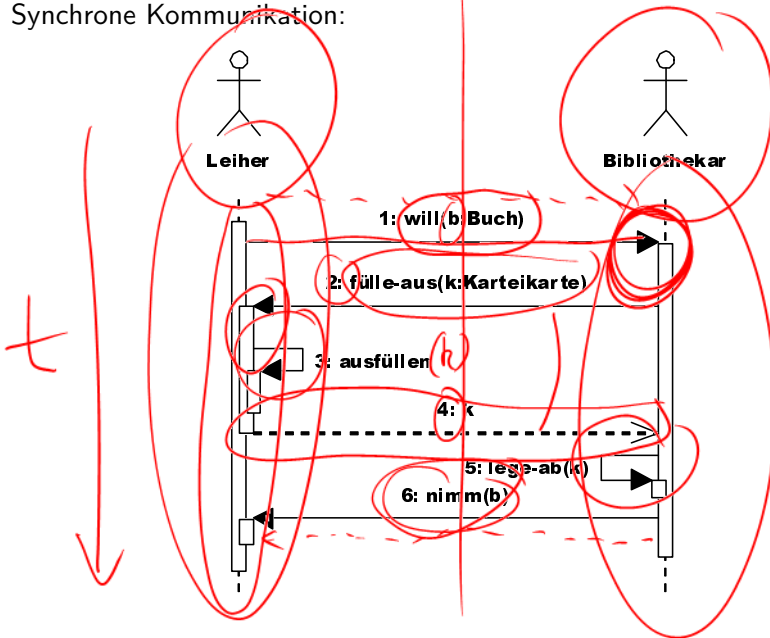
- Sequenzdiagramme
  - betonen zeitlichen Ablauf
- Kommunikationsdiagramme
  - betonen Objektstruktur

## Grenzen:

- beschreiben Verhalten meist nur beispielhaft und unvollständig (nur Nachrichtenverkehr, nicht Inhalt)

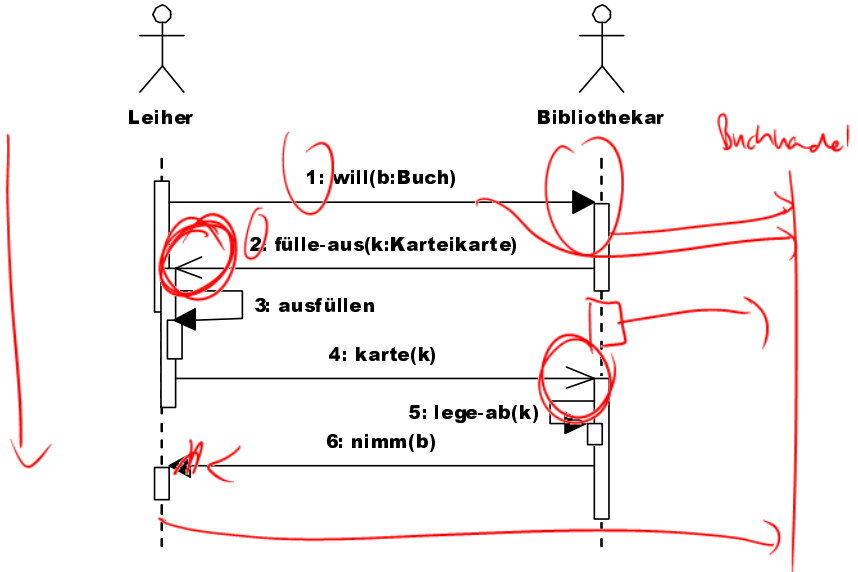
# UML-Sequenzdiagramme (OMG)

Synchrone Kommunikation:

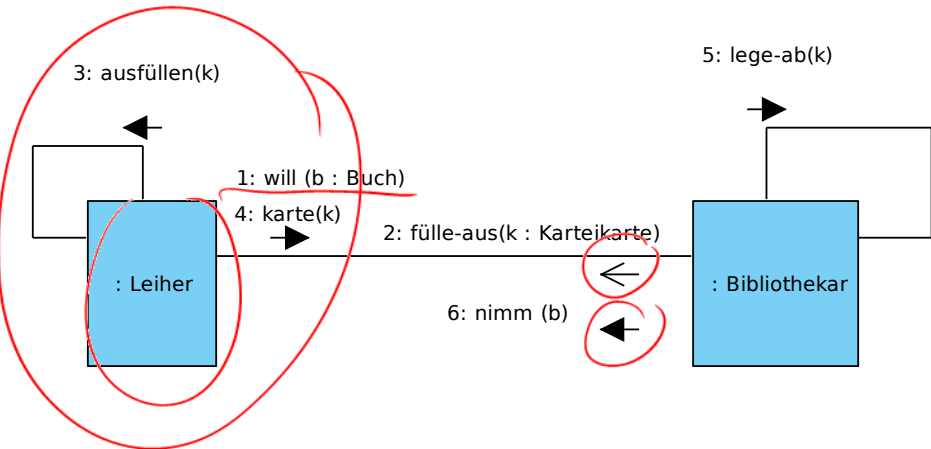


# UML-Sequenzdiagramme (OMG)

Asynchrone Kommunikation:



# UML-Kommunikationsdiagramme (OMG)

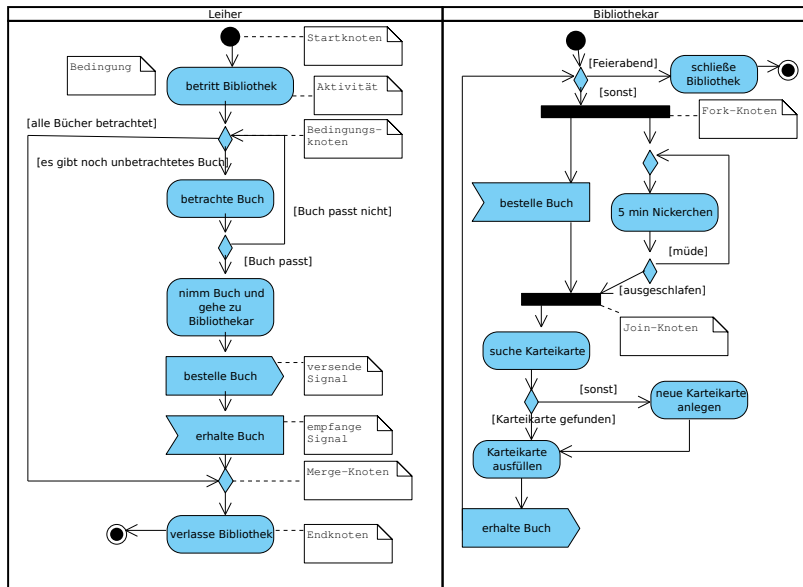


# Aktivitätsdiagramme

- Wurzeln: Flussdiagramme, Petrinetze
- modellieren klassenübergreifendes Verhalten (Kontrollfluss)
- beschreiben häufig Anwendungsfälle
- Einsatz empfohlen bei hauptsächlich intern ausgelösten Zustandsübergängen (einfacher Kontrollfluss)



# Geschäftsprozessmodellierung mit Aktivitätsdiagramm



# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

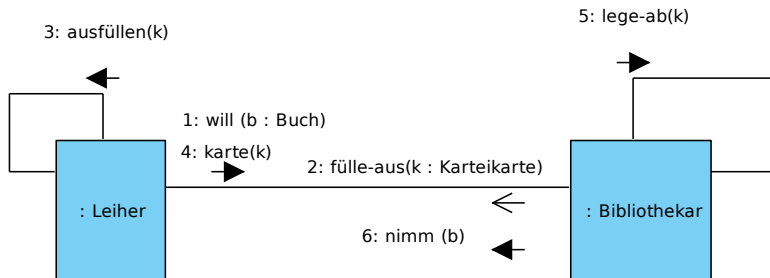
- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - ▶ Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - ▶ **Identifiziere Verhalten der Objekte**
  - ▶ Beschreibe das Verhalten (Vor- und Nachbedingungen)

# Interaktions- und Klassendiagramme

## Bemerkung:

- Interaktionsdiagramme beschreiben exemplarische Interaktionsszenarien eines Systems aus dynamischer Sicht
- Klassendiagramme beschreiben diese Systeme aus statischer, schematischer Sicht
- Interaktionsdiagramme dienen als Ausgangspunkt zur Ergänzung und Überprüfung von Klassendiagrammen

# Interaktionen → Methoden



Leiherr
+fülle_aus(k : Karteikarte) +nimm(b : Buch) -ausfüllen(k : Karteikarte)

kommunizieren

Bibliothekar
+will(b : Buch) +karte(k : Karteikarte) -lege_ab(k : Karteikarte)

# UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse

# UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse
- gehen zurück auf Zustandsautomaten nach Harel (1987)

# UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse
- gehen zurück auf Zustandsautomaten nach Harel (1987)
- **Zustand**: Menge von Attributwerten eines Objekts

# UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse
- gehen zurück auf Zustandsautomaten nach Harel (1987)
- Zustand: Menge von Attributwerten eines Objekts
- **Transition**: Zustandsänderung



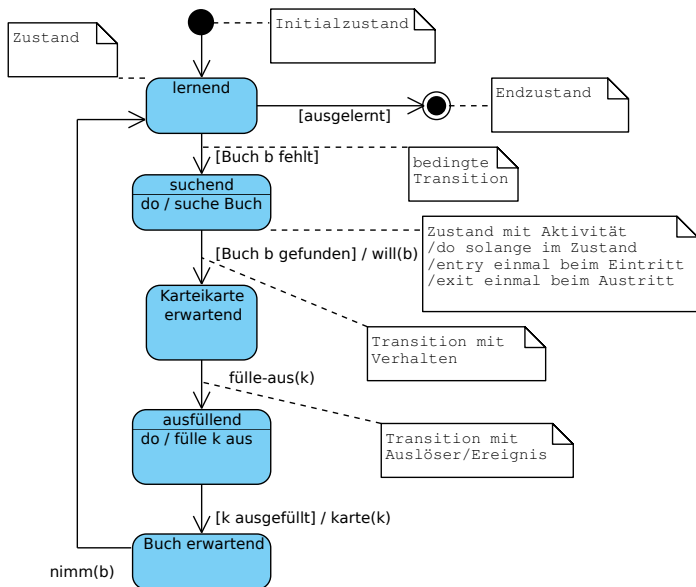
# UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse
- gehen zurück auf Zustandsautomaten nach Harel (1987)
- Zustand: Menge von Attributwerten eines Objekts
- Transition: Zustandsänderung
- geeignet auch zur Beschreibung von Anwendungsfällen und Protokollen

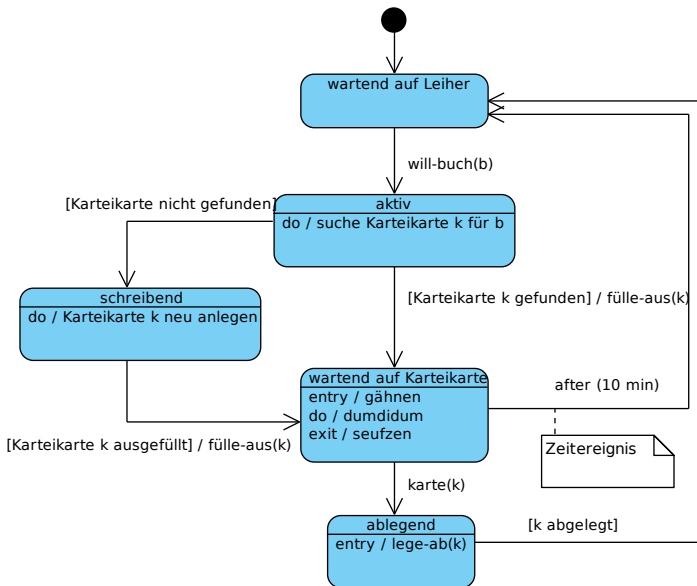
# UML-Zustandsautomatendiagramme

- modellieren Objektlebenszyklus: Verhalten eines Objekts in Bezug auf die verfügbaren Operationen seiner Klasse
- gehen zurück auf Zustandsautomaten nach Harel (1987)
- Zustand: Menge von Attributwerten eines Objekts
- Transition: Zustandsänderung
- geeignet auch zur Beschreibung von Anwendungsfällen und Protokollen
- erlauben Verschachtelung von **Zuständen** und **Automaten**

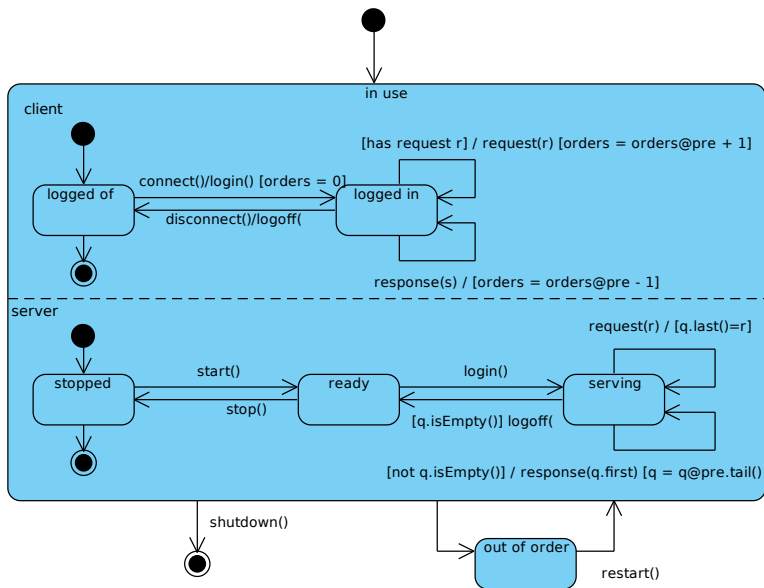
# Zustandsautomaten für Ausleiher



# Zustandsautomaten für Bibliothekar



# Zustandsautomaten für Protokolle



```
1 class Server {
2
3     private enum States {initial, stopped, ready, serving};
4     private States state;
5     private Queue requests;
6
7     public Server() {state = States.stopped; }
8     public void start() throws Exception {
9         if (state == States.stopped)
10             state = States.ready;
11         else throw new Exception();
12     }
13     public void stop() throws Exception {
14         if (state == States.ready)
15             state = States.stopped;
16         else throw new Exception();
17     }
18     public void login() throws Exception {
19         if (state == States.ready)
20             state = States.serving;
21         else throw new Exception();
22     }
23     void logoff() throws Exception {
24         if (state == States.serving)
25             state = States.ready;
26         else throw new Exception();
27     }
28     public void request(int r) throws Exception {
29         if (state == States.serving) {
30             requests.add(r); assert (requests.last() == r);
31         } else throw new Exception();
32     }
33
34 }
```

```
35 public int response() throws Exception {
36     if (state == States.serving) {
37         if (!requests.isEmpty()) {
38             int r = requests.first(); requests = requests.tail();
39             return r;
40         } else throw new Exception();
41     } else throw new Exception();
42 }
43 };
```

# Anwendung von Zustandsdiagrammen

- Beschreibung von Objektlebenszyklen pro Klasse
- Beschreibung von Protokollen
- Verfeinerung von Anwendungsfällen



# Interaktionen versus Methoden

## Interaktionen

- beschreiben das Wechselspiel zwischen Akteuren über Botschaften/Methoden<sup>1</sup>
- liefern Methoden für die modellierten Klassen
- beschreiben jedoch **nicht** die Methoden selbst

---

<sup>1</sup>Methoden beschreiben hier abstraktes Verhalten im Kontext der Anforderungen; sind noch nicht notwendigerweise die Methoden im Sinne der Implementierung (führen letztlich aber zu diesen hin).

# Objektorientierte Analyse und Modellierung

Ausgehend von Geschäftsprozessen...

- Identifiziere Akteure
- Beschreibe Anwendungsfälle
- Bestimme statisches Modell
  - ▶ Identifiziere Objekte
  - ▶ Identifiziere Eigenschaften der Objekte
  - ▶ Bestimme Assoziationen zwischen Objekten
  - ▶ Fasse Objekte zu Klassen zusammen
  - ▶ Bestimme Multiplizitäten der Assoziationen
  - ▶ Ordne Klassen in Vererbungshierarchien ein
- Erstelle Verhaltensmodell
  - ▶ Identifiziere Ereignisse und modelliere Interaktionen in Anwendungsfällen
  - ▶ Identifiziere Verhalten der Objekte
  - ▶ **Beschreibe das Verhalten (Vor- und Nachbedingungen)**

# Operationen/Nachrichten

Beschreibung:

$\subseteq_A \quad \forall 1 \leq i \leq n-1 \quad a_i \subseteq_A a_{i+1}$   
Sort(F) F[a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub>, ..., a<sub>n</sub>]

- **Parameter:** Eingabe und Ausgabe
- **Vorbedingung:** beschreibt die Annahmen der Methode, die gelten müssen, damit sie ausgeführt werden kann
- **Nachbedingung:** beschreibt das Resultat der Methode
- **Fehlerbedingungen:** Verletzung der Vorbedingungen und Fehler, die während der Ausführung auftreten können
- **Verhalten in Fehlersituationen:** Nachbedingung für jeden aufgetretenen Fehler
- **Reaktionszeit:** Maximale Dauer, bis Resultat vorliegt (sowohl im Normal- als auch im Fehlerfall)

return<sup>new</sup> LinkedList(...); ✓

# Operationen/Nachrichten

## Beispiel Sortierung und Ausgabe der Buchliste:

- **Parameter:**

- ▶ Eingabe: Buchliste, Attribut
- ▶ Ausgabe: Buchliste'

- **Vorbedingung:**

- ▶ Attribut kommt bei allen Büchern der Buchliste vor

- **Nachbedingung:**

- ▶ Buchliste' ist sortiert, d.h.: ✓  
 $\forall i : 1 \leq i < \text{len}(\text{Buchliste}') \Rightarrow \text{element}(\text{Buchliste}', i) \leq_{\text{Attribut}} \text{element}(\text{Buchliste}', i + 1)$
- ▶ Buchliste' ist eine Permutation von Buchliste ✓

- **Fehlerbedingungen:** keine, außer Vorbedingung nicht erfüllt

- **Verhalten in Fehlersituationen:**

- ▶ Buchliste wird zurückgegeben
- ▶ Fehlermeldung wird ausgegeben

- **Reaktionszeit:**  $n \cdot \log(n) \cdot 0,001$  [sec], wobei  $n$  die Länge der Liste ist ✓

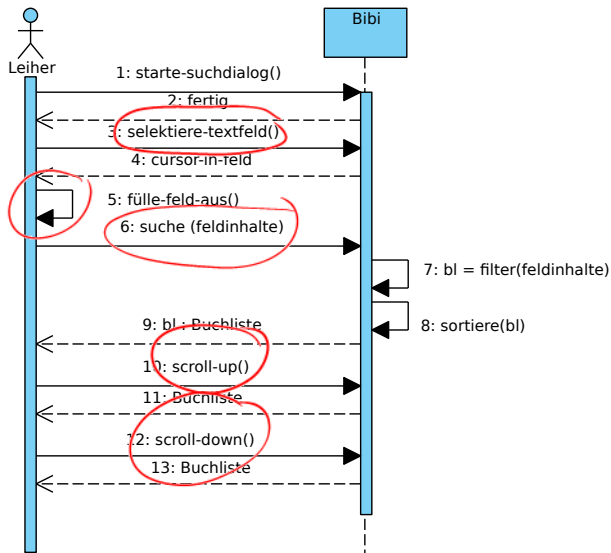


Wie lässt sich graphische Benutzeroberfläche modellieren?

Aspekte:

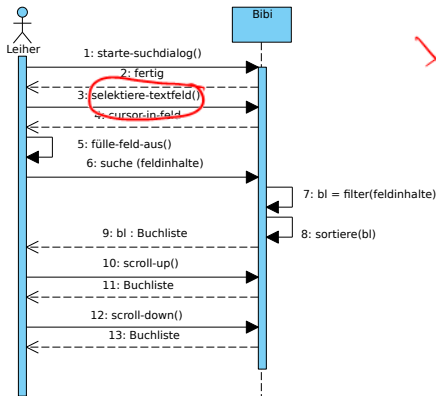
- Verhalten
- Darstellung

# Interaktion zwischen Benutzer und System (Dialog)



# Zusammenhang von Benutzernachrichten und GUI

Nachrichten vom Benutzer: Ereignisse der Eingabegeräte



selektiere-textfeld ()

linker Mausklick in Textfeld

fülle-feld-aus ()

Tastatureingabe

suche (feldinhalte)

linker Mausklick auf Schaltfläche  
"Buch suchen"

scroll-up ()

Scrollbar nach oben schieben  
oder Mausrad nach vorne

scroll-down ()

Scrollbar nach unten schieben  
oder Mausrad nach unten

# Modellierung der Darstellung

Nachrichten an Benutzer: Anzeige in GUI (hier: Suchdialog)

Buchtitel: Applied Software Architecture 1  
Autor(en): Christine Hofmeister, Robert Nord, Dilip Soni 2  
ISBN: 0-201-325 3  
Verlag: 4  
Jahr: 5  
Auflage: 6  
Band: 7  
Reihe: 8  
Besitzer: 9  
Standort: 10  
Anzahl: 11

12 Abbrechen 13 Buch suchen

1-11 Buchdaten (10pt-Schrift)

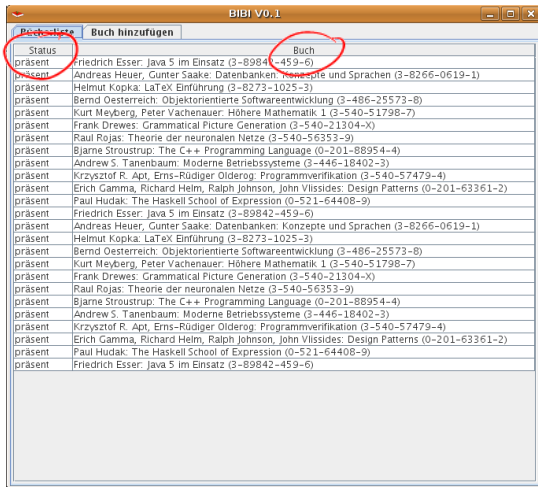
12 Schaltfläche fürs Abbrechen

13 Schaltfläche fürs Starten der Suche



# Modellierung der Darstellung

## Resultat der Suche



The screenshot shows a window titled "BIBI V0.1" with two tabs: "Buchliste" (selected) and "Buch hinzufügen". Below the tabs is a table with two columns: "Status" and "Buch". The "Status" column contains the word "präsent" for every row. The "Buch" column contains a list of books with their authors and identifiers. The first row is circled in red, and the "Status" and "Buch" headers are also circled in red.

Status	Buch
präsent	Friedrich Esser: Java 5 im Einsatz (3-89842-459-6)
präsent	Andreas Heuer, Gunter Saake: Datenbanken: Konzepte und Sprachen (3-8266-0619-1)
präsent	Helmut Kopka: LaTeX Einführung (3-8273-1025-3)
präsent	Bernd Oesterreich: Objektorientierte Softwareentwicklung (3-486-25573-8)
präsent	Kurt Meyberg, Peter Vachenaue: Höhere Mathematik 1 (3-540-51798-7)
präsent	Frank Drewes: Grammatical Picture Generation (3-540-21304-X)
präsent	Raul Rojas: Theorie der neuronalen Netze (3-540-56353-9)
präsent	Bjarne Stroustrup: The C++ Programming Language (0-201-88954-4)
präsent	Andrew S. Tanenbaum: Moderne Betriebssysteme (3-446-18402-3)
präsent	Krzysztof R. Apt, Erns-Rüdiger Olderog: Programmverifikation (3-540-57479-4)
präsent	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns (0-201-63361-2)
präsent	Paul Hudak: The Haskell School of Expression (0-521-64408-9)
präsent	Friedrich Esser: Java 5 im Einsatz (3-89842-459-6)
präsent	Andreas Heuer, Gunter Saake: Datenbanken: Konzepte und Sprachen (3-8266-0619-1)
präsent	Helmut Kopka: LaTeX Einführung (3-8273-1025-3)
präsent	Bernd Oesterreich: Objektorientierte Softwareentwicklung (3-486-25573-8)
präsent	Kurt Meyberg, Peter Vachenaue: Höhere Mathematik 1 (3-540-51798-7)
präsent	Frank Drewes: Grammatical Picture Generation (3-540-21304-X)
präsent	Raul Rojas: Theorie der neuronalen Netze (3-540-56353-9)
präsent	Bjarne Stroustrup: The C++ Programming Language (0-201-88954-4)
präsent	Andrew S. Tanenbaum: Moderne Betriebssysteme (3-446-18402-3)
präsent	Krzysztof R. Apt, Erns-Rüdiger Olderog: Programmverifikation (3-540-57479-4)
präsent	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns (0-201-63361-2)
präsent	Paul Hudak: The Haskell School of Expression (0-521-64408-9)
präsent	Friedrich Esser: Java 5 im Einsatz (3-89842-459-6)

# Modellierung von Benutzerschnittstellen

## Darstellung

- Papierskizzen
- Bilder
- Screenshots
- Animierte Darstellungen

## Verhalten

- Interaktionsdiagramme
- Zustandsdiagramme, z.B.:
  - ▶ Dialoge → zusammengesetzte Zustände
  - ▶ Interna der Dialoge → Interna in zusammengesetzten Zuständen
  - ▶ Übergänge zwischen Dialogen → Transitionen zwischen zusammengesetzten Zuständen

Interaktiver Prototyp beinhaltet all dies als Referenzimplementierung.

## Weiterführende Literatur/Links

Buchtipps: Störrle (2005) und Rupp u. a. (2007)

Ein kurzes Tutorial in Deutsch:

<http://ivs.cs.uni-magdeburg.de/~dumke/UML/>

Eine sehr kurze Übersicht in Englisch:

<http://bdn.borland.com/article/0,1410,31863,00.html>

Weitere ausführlichere Tutorials in Englisch:

[http:](http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/)

[//pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML\\_tutorial/](http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/)

<http://uml.tutorials.tireme.com/>